Semantic Theory

Week 2: Type Theory

Noortje Venhuizen & Harm Brouwer — Universität des Saarlandes — Summer 2022

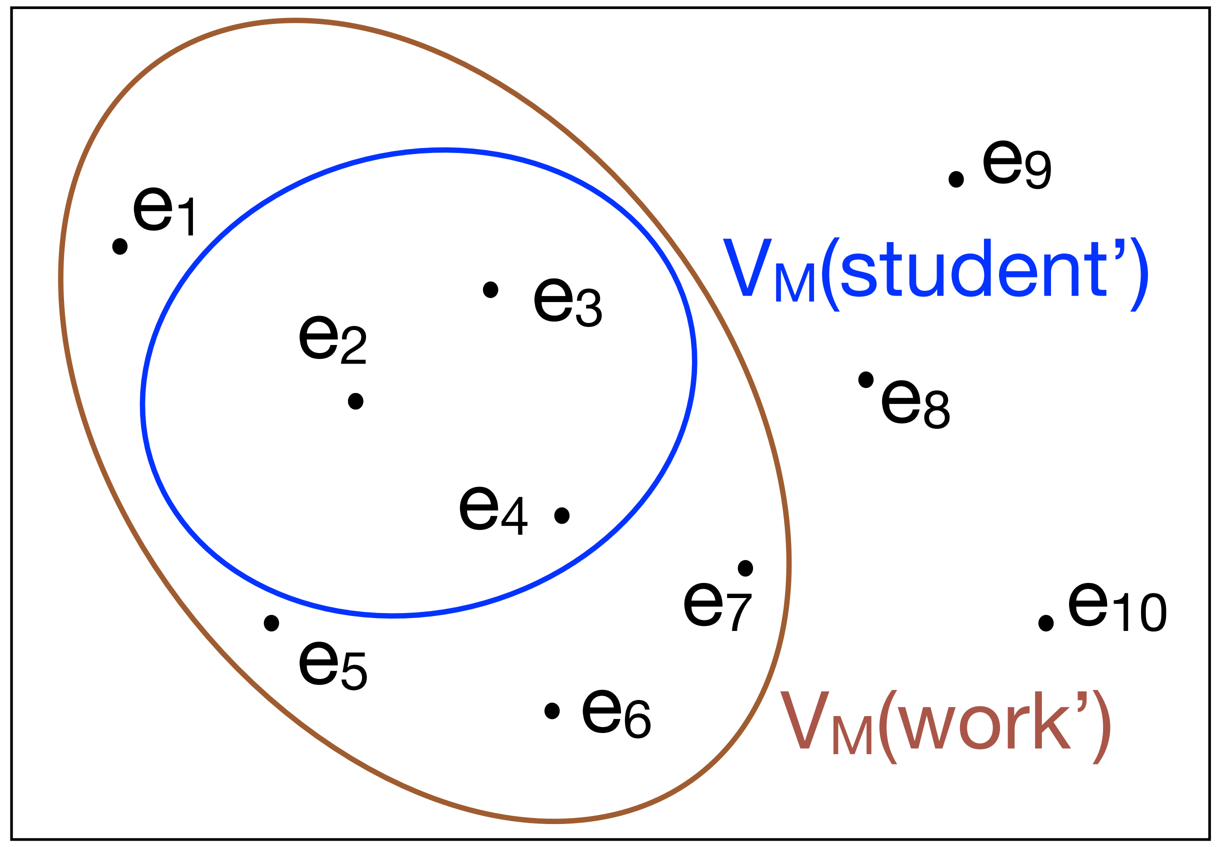
Truth-conditional semantics

Assumption: Logical formula captures truth-conditions of NL sentence; they are true in the same possible models.

“John is a student.” ⟺ student’(john’)

natural language *logic*

true ⇔ true



set theory

Truth, validity and entailment

- A formula φ is true in a model M iff:
  $[\![\varphi]\!]^{M,g} = 1$ for every variable assignment g

- A formula φ is valid ($\vDash \varphi$) iff:
  φ is true in all models

- A formula φ is satisfiable iff:
  there is at least one model M such that φ is true in M

- A set of formulas Γ entails formula φ ($\Gamma \vDash \varphi$) iff:
  φ is true in every model in which all formulas in Γ are true
  - the elements of Γ are called the premises or hypotheses
  - φ is called the conclusion

First-order logic

Predication and quantification over individual entities

- First-order logic talks about:
  - Individual objects: $V_M(john') \in U_M$; $g(x) \in U_M$
  - Properties of and relations between individual objects: happy'(john'); love'(john',mary')
  - Quantification over individual objects: $\forall x(happy(x))$

Limitations of first-order logic

FOL is not expressive enough to capture all meanings that can be expressed by basic natural language expressions:

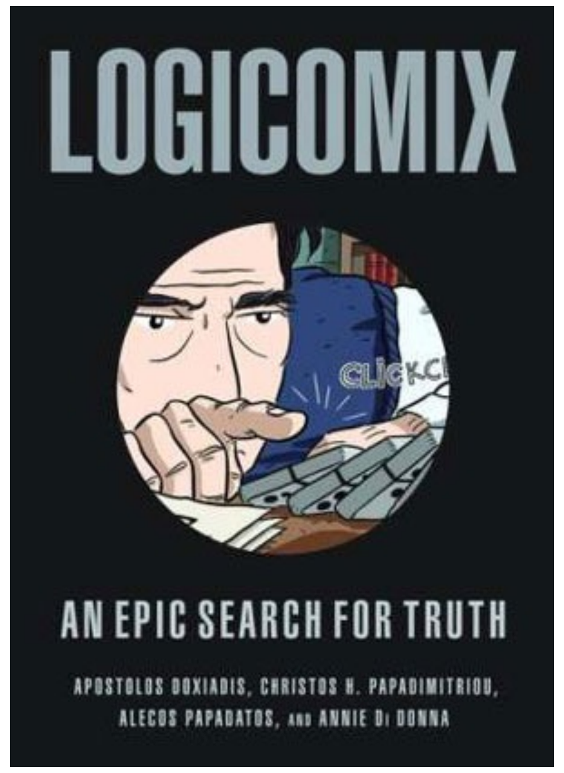
- Jumbo is a <u>small</u> elephant. (Predicate modifiers)
- Being rich is a <u>state of mind.</u> (Second-order predicates)
- <u>Yesterday</u>, it rained. (Non-logical sentence operators)
- Bill and John have <u>the same</u> hair color. (Higher-order quantification)

➔ What system can capture this diversity?

Simple idea: introduce higher order predication & quantification

Introducing Russell's paradox

Bertrand Russell



From: **Logicomix — An epic search for truth**; A. Doxiadis, C.H. Papadimitriou, A. Papadatos and A. Di Donna

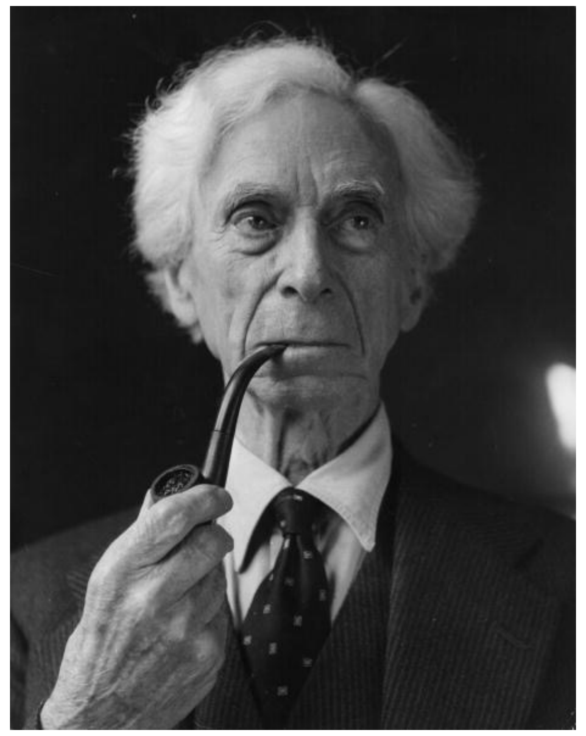
Problem for higher-order predicate logic

Russell's paradox

What if we extend the FOL interpretation of predicates, and simply interpret higher-order predicates as sets of sets of properties?

- For every predicate P, we define a set {x | P(x)} containing all and only those entities for which P holds; higher order predicates are defined as sets of sets, e.g., {P | H(P)}
- This means that we can formally define a set S = {X | X ∉ X} representing the set of all sets that are not members of itself
- **Paradox**: does S belong to itself?

  If it does, then S must satisfy its constraints, namely that it doesn't belong to itself, which is not possible if we assume it belongs to S. If not, then S is a set that doesn't belong to itself, hence it belongs to S.



Bertrand Russell
1872 — 1970

→ Conclusion: We need a more restricted way of talking about *properties* and *relations between properties*!

Type Theory

Basic and complex types

Winter: EFS Ch3
Page 50

- In Type Theory, all non-logical expressions are assigned a *type* (that may be basic or complex), which restricts how they can be combined.
- Basic types:
  - **e** – the type of individual terms (“entities”)
  - **t** – the type of formulas (“truth-values”)

✅ ❌

- Complex types:
  - If **π, σ** are types, then ⟨**π,σ**⟩ is a type

    This represents a functor expression that takes an expression of type **π** as its argument and returns an expression of type **σ**; this functor is sometimes written as **(π → σ)** or simply **(πσ)**

Alonzo Church
1903 — 1995

Type Theory

Types & Function Application

Winter: EFS Ch3 Page 53

Types for first-order expressions:

- Individual constants (Luke, Death Star) : **e** **(Entity)**
- One-place predicates (to walk, to be a jedi): ⟨**e, t**⟩ **(Function from entities to truth values; a property)**
- Two-place predicates (to admire, to fight with): ⟨**e,** ⟨**e, t**⟩⟩ **(Function from entities to properties)**
- Three-place predicates (to give, to introduce): ⟨**e,** ⟨**e,** ⟨**e, t**⟩⟩⟩ **(Function from entities to functions from entities to properties)**

Function application: Combining a functor of complex type ⟨**π, σ**⟩ with an appropriate argument of type **π**, results in an expression of type **σ**: ⟨**π, σ**⟩(**π**) ↦ **σ**

- jedi'(luke') :: ⟨**e, t**⟩(**e**) ↦ **t** **("luke is a jedi": statement that has a truth value)**
- admire'(luke') :: ⟨**e,**⟨**e, t**⟩⟩(**e**) ↦ ⟨**e, t**⟩ **("[to] admire luke" is a property)**

More examples of types

Higher-order expressions

- Predicate modifiers (expensive, small): ⟨⟨**e, t**⟩**,** ⟨**e, t**⟩⟩ **(Function from properties to properties)**
- Second-order predicates (state of mind): ⟨⟨**e, t**⟩**, t**⟩ **(Property of properties)**
- Sentence operators (yesterday, unfortunately): ⟨**t, t**⟩ **(Function from truth values to truth values)**
- Degree particles (very, too): ⟨⟨⟨**e, t**⟩**,** ⟨**e, t**⟩⟩**,** ⟨⟨**e, t**⟩**,** ⟨**e, t**⟩⟩⟩ ***complex function***

If **π, σ** are basic types, ⟨**π, σ**⟩ can be abbreviated as **πσ**. The types of predicate modifiers and second-order predicates can then be more conveniently written as: ⟨**et, et**⟩ and ⟨**et, t**⟩.

Type Theory: Vocabulary

- **Non-logical constants:**

  A (possibly empty) set of non-logical constants for every type σ: CON_σ
  such that the sets for all distinct types are pairwise disjoint

- **Variables:**

  An infinite set of variables For every type σ: VAR_σ (pairwise disjoint)

- **Logical symbols**: ∀, ∃, ¬, ∧, ∨, →, ↔, =

- **Brackets:** (,)

Type Theory: Syntax

For every type σ, the set of well-formed expressions WE_σ is defined as follows:

(i) $CON_\sigma \subseteq WE_\sigma$ and $VAR_\sigma \subseteq WE_\sigma$;

(ii) If $\alpha \in WE_{\langle\pi, \sigma\rangle}$, and $\beta \in WE_\pi$, then $\alpha(\beta) \in WE_\sigma$; (function application)

(iii) If A, B are in WE_t, then $\neg A$, $(A \wedge B)$, $(A \vee B)$, $(A \rightarrow B)$, $(A \leftrightarrow B)$ are in WE_t;

(iv) If A is in WE_t and x is a variable of arbitrary type, then $\forall xA$ and $\exists xA$ are in WE_t;

(v) If α, β are well-formed expressions of the same type, then $\alpha = \beta \in WE_t$;

(vi) Nothing else is a well-formed expression.*

**NB: This prevents us from running into Russell's paradox!*

Type inferencing

Winter: EFS Ch3
Page 59-60

- Based on the syntactic structure of a sentence, we can derive its logical form, which defines how functions and arguments are combined
- Each expression that constitutes the logical form obtains a type, which can be inferred from the function-argument structure
- Luke is a talented jedi ⇔* talented’(jedi’)(luke’)

$$\dfrac{\text{luke'} :: e \qquad \dfrac{\text{talented} :: \langle\langle e, t\rangle, \langle e, t\rangle\rangle \qquad \text{jedi'} :: \langle e, t\rangle}{\text{talented'(jedi')} :: \langle e, t\rangle}}{\text{talented'(jedi')(luke')} :: t}$$

* Note: we here ignore the semantic contribution of “is” and “a” (see Winter, pg 61)

Type inferencing: examples

Recommended strategy: Start by describing the logical form of the sentences (how are functions and arguments combined, based on the given syntactic bracketing), then derive types for all relevant sub-expressions (see previous slide).

1. Yoda$_e$ [<u>is faster than</u> Palpatine$_e$].

2. Yoda$_e$ [is <u>much</u> [faster than]] Palpatine$_e$.

3. [[Han Solo]$_e$ fights] [<u>because</u> [[the Dark Side]$_e$ is rising]].

4. Obi-Wan$_e$ [[<u>told</u> [Qui-Gon Jinn]$_e$] he will take [the Jedi-exam]$_e$].

Higher-order predicates

Higher-order quantification:

- *Leia has the same hair colour as Padmé*

$$\exists C\ (hair\_colour(C) \wedge C(l') \wedge C(p'))$$

hair_colour: $\langle\langle e, t\rangle, t\rangle$ — C: $\langle e, t\rangle$ — l': e

Higher-order equality:

- For p, q ∈ CON_t, "p=q" expresses material equivalence: "p ↔ q".
- For F, G ∈ $CON_{\langle e, t\rangle}$, "F=G" expresses co-extensionality: "∀x(Fx↔Gx)"
- For any formula ϕ of type *t*, ϕ=(x=x) is a representation of "ϕ is true".

Type Theory: Semantics

Winter: EFS Ch3
Page 51

Type domains

- Let **U** be a non-empty set of entities.
- The domain of possible denotations $\mathbf{D_\sigma}$ for every type $\boldsymbol{\sigma}$ is given by:
  - $D_e = U$
  - $D_t = \{0,1\}$
  - $D_{\langle\pi,\,\sigma\rangle}$ is the set of all functions from D_π to D_σ: $D_\sigma^{D_\pi}$
- For any type $\boldsymbol{\sigma}$, expressions of type $\boldsymbol{\sigma}$ denote elements of the domain $\mathbf{D_\sigma}$

Type Theory: Semantics

Example domains

- For M = ⟨U, V⟩, let U consist of five entities. For selected types, we have the following sets of possible denotations:
  - $D_t = \{0,1\}$
  - $D_e = U = \{e_1, e_2, e_3, e_4, e_5\}$
  - $D_{<e,t>} = \{ \begin{bmatrix} e_1 \to 0 \\ e_2 \to 0 \\ e_3 \to 0 \\ e_4 \to 0 \\ e_5 \to 0 \end{bmatrix}, \begin{bmatrix} e_1 \to 1 \\ e_2 \to 0 \\ e_3 \to 0 \\ e_4 \to 0 \\ e_5 \to 0 \end{bmatrix}, \ldots, \begin{bmatrix} e_1 \to 0 \\ e_2 \to 1 \\ e_3 \to 1 \\ e_4 \to 0 \\ e_5 \to 1 \end{bmatrix}, \ldots, \begin{bmatrix} e_1 \to 1 \\ e_2 \to 1 \\ e_3 \to 0 \\ e_4 \to 0 \\ e_5 \to 1 \end{bmatrix}, \ldots \}$

Equivalent set notation: $D_{<e,t>} = \{\{\},\{e_1\},\ldots,\{e_2,e_3,e_5\},\ldots,\{e_1,e_2,e_5\},\ldots\}$

Characteristic functions

Winter: EFS Ch3 Page 46-47

- Many natural language expressions have a type ⟨**σ, t**⟩; expressing functions that map elements of type **σ** to truth values: {**0,1**}

- Such functions with a range of {**0,1**} are called *characteristic functions*, because they uniquely specify a subset of their domain $\mathbf{D_\sigma}$

The characteristic function of set S in a domain U is the function $F_S: U \rightarrow \{0,1\}$ such that for all $e \in U$, $F_M(e) = 1$ iff $e \in S$.

- NB: For first-order predicates, the FOL denotation (using sets) and the type-theoretic denotation (using characteristic functions) are equivalent.

Type Theory: Semantics

Model-theoretic interpretation

- A model structure for a type theoretic language is a tuple **M = ⟨U, V⟩** such that:
  - **U** is a non-empty domain of individuals
  - **V** is an interpretation function, which assigns to every $\boldsymbol{\alpha} \in \mathbf{CON}_{\boldsymbol{\sigma}}$ an element of $\mathbf{D}_{\boldsymbol{\sigma}}$ (where $\boldsymbol{\sigma}$ is an arbitrary type)
- The variable assignment function **g** assigns to every typed variable $\mathbf{v} \in \mathbf{VAR}_{\boldsymbol{\sigma}}$ an element of the domain $\mathbf{D}_{\boldsymbol{\sigma}}$ (where $\boldsymbol{\sigma}$ is an arbitrary type): $\mathbf{g} :: \mathbf{VAR}_{\boldsymbol{\sigma}} \rightarrow \mathbf{D}_{\boldsymbol{\sigma}}$
- **Interpretation of expressions**: Given model structure M = ⟨U, V⟩ and assignment g:
  - $[\![\alpha]\!]^{M,g} = V(\alpha)$ if α is a constant
    $= g(\alpha)$ if α is a variable
  - $[\![\alpha(\beta)]\!]^{M,g} = [\![\alpha]\!]^{M,g}([\![\beta]\!]^{M,g})$ *function application*

Type Theory: Semantics

Interpretation of formulas

- Given a type-theoretic model structure **M = ⟨U, V⟩** and variable assignment **g**:
  - $[\![\alpha = \beta]\!]^{M,g} = 1$ iff $[\![\alpha]\!]^{M,g} = [\![\beta]\!]^{M,g}$
  - $[\![\neg\phi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 0$
  - $[\![\phi \wedge \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 1$ and $[\![\psi]\!]^{M,g} = 1$
  - $[\![\phi \vee \psi]\!]^{M,g} = 1$ iff $[\![\phi]\!]^{M,g} = 1$ or $[\![\psi]\!]^{M,g} = 1$
  - …
- For any variable **v** of type **σ**:
  - $[\![\exists v\phi]\!]^{M,g} = 1$ iff there is a $d \in D_\sigma$ such that $[\![\phi]\!]^{M,g[v/d]} = 1$
  - $[\![\forall v\phi]\!]^{M,g} = 1$ iff for all $d \in D_\sigma$: $[\![\phi]\!]^{M,g[v/d]} = 1$

Type-theoretic interpretation

Example

Luke is a talented jedi ⇔ $\text{talented'}_{\langle\langle e,t\rangle,\langle e,t\rangle\rangle}(\text{jedi'}_{\langle e,t\rangle})(\text{luke'}_e)$

$$[\![\text{talented'}(\text{jedi'})(\text{luke'})]\!]^{M,g} = [\![\text{talented'}(\text{jedi'})]\!]^{M,g}([\![\text{luke'}]\!]^{M,g})$$

$$= [\![\text{talented'}]\!]^{M,g}([\![\text{jedi'}]\!]^{M,g})([\![\text{luke'}]\!]^{M,g}) = \underline{V_M(\text{talented'})(V_M(\text{jedi'}))}\underline{(V_M(\text{luke'}))}$$

$$\begin{bmatrix} e_1 \to 1 \\ e_2 \to 0 \\ e_3 \to 0 \\ e_4 \to 0 \\ e_5 \to 1 \end{bmatrix}(e_1) = \mathbf{1}$$

Assume the following denotations:

- $V_M(\text{luke'}) = e_1 \; (\in \mathbf{D_e})$
- $V_M(\text{jedi'}) = \begin{bmatrix} e_1 \to 1 \\ e_2 \to 0 \\ e_3 \to 1 \\ e_4 \to 0 \\ e_5 \to 1 \end{bmatrix} \; (\in \mathbf{D_{\langle e,t\rangle}})$
- $V_M(\text{talented'}) = \begin{bmatrix} \begin{bmatrix} e_1 \to 1 \\ e_2 \to 0 \\ e_3 \to 1 \\ e_4 \to 0 \\ e_5 \to 1 \end{bmatrix} \to \begin{bmatrix} e_1 \to 1 \\ e_2 \to 0 \\ e_3 \to 0 \\ e_4 \to 0 \\ e_5 \to 1 \end{bmatrix}, \begin{bmatrix} e_1 \to 0 \\ e_2 \to 0 \\ e_3 \to 1 \\ e_4 \to 1 \\ e_5 \to 1 \end{bmatrix} \to \begin{bmatrix} e_1 \to 0 \\ e_2 \to 0 \\ e_3 \to 0 \\ e_4 \to 0 \\ e_5 \to 1 \end{bmatrix}, \ldots \end{bmatrix} \; (\in \mathbf{D_{\langle\langle e,t\rangle\langle e,t\rangle\rangle}})$

Type-theoretic models of natural language

Defining the right model

Consider the following Model M:

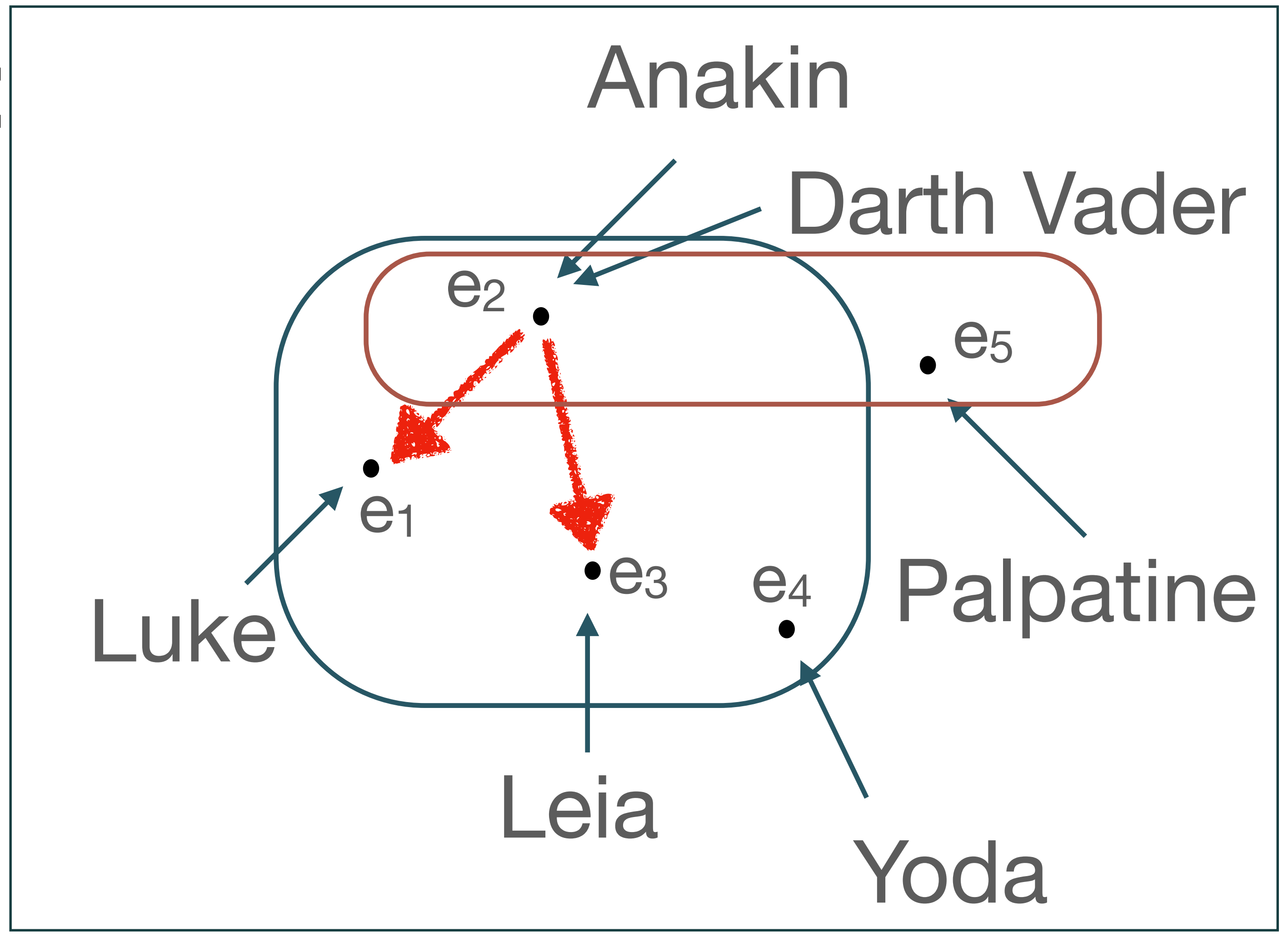
$D_e = U_M = \{e_1, e_2, e_3, e_4, e_5\}$

$V_M(\text{anakin'}_e) = V_M(\text{darth\_vader'}_e) = e_2$

$V_M(\text{jedi'}_{\langle e,t \rangle}) = \begin{bmatrix} e_1 \rightarrow 1 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 1 \\ e_4 \rightarrow 1 \\ e_5 \rightarrow 0 \end{bmatrix}$ $\quad V_M(\text{dark\_sider'}_{\langle e,t \rangle}) = \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 0 \\ e_5 \rightarrow 1 \end{bmatrix}$

$$V_M(\text{powerful'}_{\langle\langle e,t \rangle \langle e,t \rangle\rangle}) = \begin{bmatrix} \begin{bmatrix} e_1 \rightarrow 1 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 1 \\ e_4 \rightarrow 1 \\ e_5 \rightarrow 0 \end{bmatrix} \rightarrow \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 1 \\ e_5 \rightarrow 0 \end{bmatrix} , \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 0 \\ e_5 \rightarrow 1 \end{bmatrix} \rightarrow \begin{bmatrix} e_1 \rightarrow 0 \\ e_2 \rightarrow 1 \\ e_3 \rightarrow 0 \\ e_4 \rightarrow 0 \\ e_5 \rightarrow 1 \end{bmatrix} , \ldots \end{bmatrix}$$

M:



Note that "powerful" is defined to be truth-preserving: Powerful $X_{\langle e,t \rangle} \vDash X_{\langle e,t \rangle}$

Meaning postulates

Restricting denotations

- Some valid inferences in natural language:
  - Bill is a poor piano player ⊨ Bill is a piano player
  - Bill is a blond piano player ⊨ Bill is blond
  - Bill is a former professor ⊨ Bill isn't a professor

➔ These entailments do not hold in type theory by definition.

Meaning postulates: Restrictions on models that constrain the possible meanings of certain words

Meaning postulates for adjective classes

- **Restrictive or subsective adjectives** (e.g., “poor”)
  - Restriction: $[\![\text{poor N}]\!] \subseteq [\![\text{N}]\!]$
  - Meaning postulate: $\forall G \forall x(\text{poor}(G)(x) \rightarrow G(x))$
- **Intersective adjectives** (e.g., “blond”)
  - Restriction: $[\![\text{blond N}]\!] = [\![\text{blond}]\!] \cap [\![\text{N}]\!]$
  - Meaning postlate: $\forall G \forall x(\text{blond}(G)(x) \rightarrow (\text{blond*}(x) \wedge G(x))$
  - NB: $\text{blond} \in WE_{\langle\langle e, t\rangle, \langle e, t\rangle\rangle} \neq \text{blond*} \in WE_{\langle e, t\rangle}$
- **Privative adjectives** (e.g.,“former”)
  - Restriction: $[\![\text{former N}]\!] \cap [\![\text{N}]\!] = \varnothing$
  - Meaning postlate: $\forall G \forall x(\text{former}(G)(x) \rightarrow \neg G(x))$

Reading material

Recommended reading

- Winter: Elements of Formal Semantics (Chapter 3, Part I & II)
  http://www.phil.uu.nl/~yoad/efs/main.html